# Práctica 4: Módulos Odoo Modelo y Vista

Eder Martínez Castro

13 de diciembre de 2025

# Índice general

# 1. Introducción

El objetivo de esta práctica es seguir trabajando con el modelo y la vista de los módulos de Odoo, para ello tendremos que implementar cuatro módulos, **Lista de tareas**, **Biblioteca de cómics**, **Pacientes y médicos**, **Ciclos formativos**.

Antes de proceder a empezar con la práctica se nos recomienda revisar los ejemplos del 01 al 06 que tenemos en el siguiente repositorio github.com/sergarb1/OdooModulosEjemplos.git.

Para ello deberemos clonarlo en nuetro equipo utilizando el siguiete comando:

```
git clone https://github.com/sergarb1/OdooModulosEjemplos.git
```

Cuando ya lo tengamos clonado lo que haremos es copiar los 6 ejemplos y pegarlos en nuestro proyecto para poder probarlos y revisarlos. Y nos debería quedar nuestra carpeta con nuestro módulos tal que así:


Figura 1: Carpeta addons, que contiene nuestros módulos de Odoo donde podemos ver los módulos de ejemplo del repositorio para probarlos.

Entender estos módulos de ejemplo nos ayudará mucho para poder hacer la actividad 1 de lista de tareas y la 2 de la biblioteca de comics.

# 2. Actividad 01 - Lista de tareas

## 2.1. Objetivo

Para esta actividad se nos pide coger el código de nuestro módulo de ejemplo 2 de la lista de tareas, para modificarlo e implementar en el que podamos verlo además del **modo lista**, poder verlo en **modo kanban** y **calendario**.

## 2.2. Agregar vista Kanban

Para poder implementar la vista estilo kanban solo necesitamos modificar el archivo `views.xml` que es nuestra vista para poder implementar este nuevo modo de visualización.

```
<!-- ================================================
     VISTA DE KANBAN
     ================================================ -->
<record id="view_lista_tareas_kanban" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.kanban</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo kanban -->
    <kanban default_order_by="realizada">
      <field name="tarea"/>
      <field name="prioridad"/>
      <field name="urgente"/>
      <field name="realizada"/>

      <templates>
        <!-- Tarjeta de las tareas de kanban -->
        <t t-name="kanban-box">
          <div class="oe_kanban_card oe_kanban_global_click">
            <strong><field name="tarea"/></strong>
            <div>
              <div>Prioridad: <field name="prioridad"/></div>
              <div>Urgente: <field name="urgente"/></div>
              <div>Realizada: <field name="realizada"/></div>
            </div>
          </div>
        </t>
      </templates>
    </kanban>
  </field>
</record>
```

Figura 2: Nuevo fragmento de código para mostrar las tareas en formato kanban.

En este fragmento de código implementamos la nueva vista tipo kanban, que nos mostrará nuestras tareas en tarjetas individuales con los datos de la tarea.

Estrucutura principal de la vista Kanban:

- `<kanban>`: muestra la vista en formato kanban como bien indica el nombre
- `<field>`: declaramos los campos que tendra la vista.
- `<templates>`: contiene la plantilla `kanban-box` donde creamos la tarjeta de la tarea del kanban:
  - La tarjeta es interactiva al usar `oe_kanban_global_click` si hacemos click en ella nos abrira el formulario para poder editarla.
  - Se muestran los datos de la tarea: nombre, prioridad, urgencia y si está o no realizada.

```
<!-- ================================================
     ACCIÓN PRINCIPAL DEL MÓDULO
     ================================================
     Esta acción es la que se ejecutará cuando el usuario
     entre en el menú "Ver tareas".
     Abre el modelo lista_tareas.lista mostrando sus vistas.
-->
<record model="ir.actions.act_window" id="action_lista_tareas">
  <!-- Título visible de la ventana -->
  <field name="name">Listado de tareas</field>

  <!-- Modelo al que se refiere la acción -->
  <field name="res_model">lista_tareas.lista</field>

  <!-- Tipo de vistas que se mostrarán:
       list → vista en tabla
       form → vista en formulario
       kanban -> vista kanban -->
  <field name="view_mode">list,form,kanban</field>
</record>
```

Figura 3: En el bloque para las acciones de nuestro módulo debemos agregar como se ve en pantalla en el `view_mode` el nuevo tipo kanban.

En esta modificación lo que hicimos fue agregar la vista kanban al `view_mode`, porque Odoo utiliza este parámetro para determinar que vistas estarán disponibles en la interfaz de nuestro módulo.

## 2.3. Comprobación de la vista Kanban

A continuación vamos a enseñar a través de capturas de pantalla el correcto funcionamiento de nuestra vista kanban.

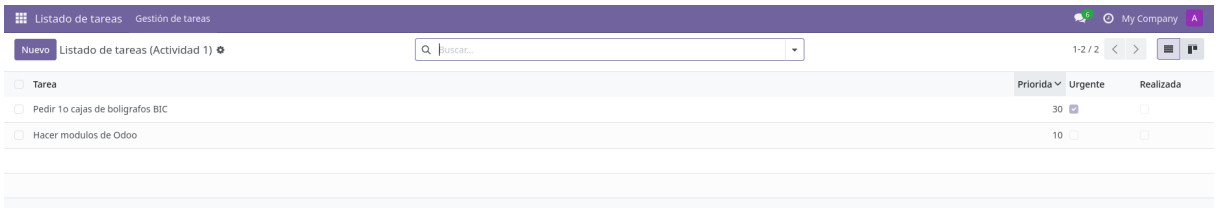



Figura 4: En esta captura podemos ver la vista modo lista de nuestro módulo de listas de tareas que es la primera que nos saldrá cuando lo abramos.

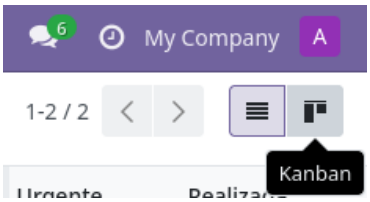



Figura 5: Como se ve en esta captura para cambiar de vista nos iremos a estos iconos y pulsaremos en el que pone kanban para poder ver nuestras tareas en este formato

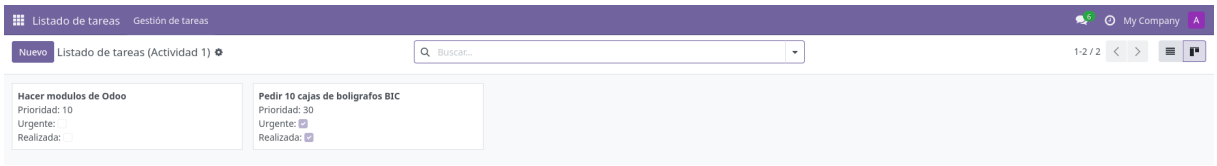



Figura 6: En esta captura podemos ver la vista modo kanban de nuestro módulo de listas de tareas.

Aquí podemos ver nuestras tareas en tarjetas con todos sus campos relevantes, lo que muestra que tenemos la vista del kanban bien implementada.

## 2.4. Agregar vista Calendar

Para poder implementar la vista estilo calendar deberemos modificar el archivo `views.xml` que es nuestra vista y el archivo `models.py` que es nuestro modelo para poder implementar este nuevo modo de visualización.

Primero vamos a empezar modificando nuestro `models.py`:

```
# Campos para la fecha de inicio y fin de la tarea
fecha_inicio = fields.Date(string="Fecha inicio")
fecha_fin = fields.Date(string="Fecha fin")
```

Figura 7: En esta captura podemos ver como hemos agregado a nuestra modelo los campos `fecha_inicio` y `fecha_fin` para poder marcar el comienzo de una tarea y su final.

Para que nuestra vista calendario pueda mostrar nuestras tareas, necesitamos crear dos nuevos campos para poder marcar la fecha de inicio de la tarea y la fecha de fin de esta.

- `fecha_inicio`: fecha en la que esta tarea empieza.
- `fecha_fin`: fecha límite de la tarea.

Ahora vamos a realizar las siguientes modificaciones en nuestro `views.xml`:

```
<!-- ====================================================
     VISTA DE CALENDARIO
     ==================================================== -->
<record id="view_lista_tareas_calendar" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.calendar</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo calendar -->
    <calendar date_start="fecha_inicio" date_stop="fecha_fin">
      <field name="tarea"/>
      <field name="prioridad"/>
      <field name="urgente"/>
      <field name="realizada"/>
    </calendar>
  </field>
</record>
```

Figura 8: Nuevo fragmento de código para mostrar las tareas en formato calendar.

En este fragmento de código implementamos la nueva vista tipo calendar, que nos mostrará un calendario con nuestras tareas desde el dia que empiezan hasta el dia que deben estar finalizadas.

Estrucutura principal de la vista calendar:

- `<calendar>`: muestra la vista en formato calendario como bien indica el nombre, esta recibe dos atributos:
  - `date_start="fecha_inicio"`: indicamos el día de comienzo de la tarea.
  - `date_start="fecha_fin"`: indicamos el día final de la tarea.
- `<field>`: declaramos los campos que tendra la vista.

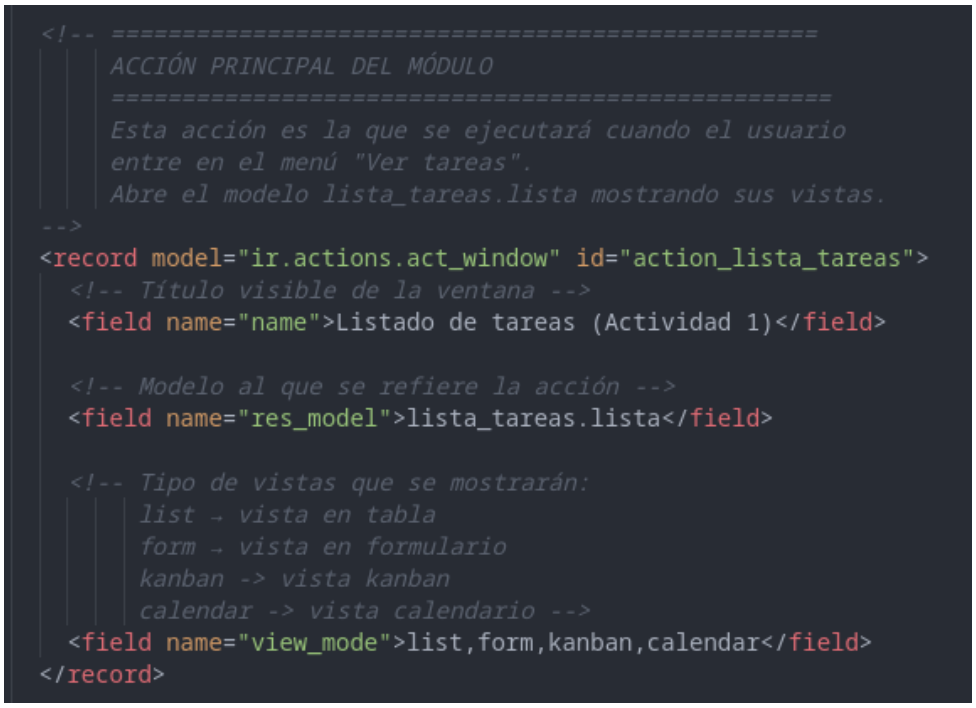



Figura 9: En el bloque para las acciones de nuestro módulo debemos agregar como se ve en pantalla en el `view_mode` el nuevo tipo calendar.

En esta modificación lo que hicimos fue agregar la vista calendar al `view_mode`, como ya explicamos anteriormente esto es porque Odoo utiliza este parámetro para determinar que vistas estarán disponibles en la interfaz de nuestro módulo.

Ahora que ya hemos implementado nuestra vista kanban en el `views.xml`, pero todavía nos falta hacer unas modificaciones en los bloques del formulario para poder agregar la fecha de inicio y fin a las tareas cuando las creemos o editemos, además de agregar estos nuevos campos de fecha a las vistas de lista y kanban:

```xml
<!-- ================================================
     VISTA DE FORMULARIO (form)
     ================================================
     Se usa al crear o editar una tarea.
     Más amigable y detallada que la vista de lista.
-->
<record id="view_lista_tareas_form" model="ir.ui.view">
  <field name="name">lista.tareas.form</field>
  <field name="model">lista_tareas.lista</field>
  <field name="arch" type="xml">
    <form string="Tarea">
      <!-- El contenedor visual principal del formulario -->
      <sheet>
        <!-- Primer bloque de campos (agrupados) -->
        <group>
          <field name="tarea"/>          <!-- Campo de nombre de tarea -->
          <field name="realizada"/>      <!-- Campo para marcar como realizada -->
        </group>

        <!-- Segundo bloque de campos (prioridad y urgencia) -->
        <group string="Prioridad y urgencia">
          <field name="prioridad"/>      <!-- Campo de prioridad editable -->

          <!-- Campo calculado: no editable. Ya no usamos attrs (Odoo 17+) -->
          <field name="urgente" readonly="1"/>
        </group>

        <!-- Tercer bloque de campos para las fechas de la tarea -->
        <group string="Fechas de la tarea">
          <field name="fecha_inicio"/>
          <field name="fecha_fin"/>
        </group>

      </sheet>
    </form>
  </field>
</record>
```

Figura 10: Actualizamos el bloque de formulario para tener un nuevo bloque para agregar las fechas de inicio y fin de la tarea para cuando agregemos una tarea o la editemos.

En el formulario, agregamos un nuevo grupo para las fechas de la tarea donde tenemos los campos de fecha de inicio y fecha final de la tarea, para cuando creemos y editemos una tarea podamos agregar las fechas.

```xml
<!-- ================================================
     VISTA DE LISTA (list)
     ================================================
     Muestra las tareas en forma de tabla.
     Es la vista que se usa por defecto para ver varios registros.
-->
<record id="view_lista_tareas_list" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.list</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo list (reemplaza a <tree> en Odoo 17+) -->
    <list string="Tareas" default_order="prioridad desc">
      <!-- Campos que se muestran como columnas -->
      <field name="tarea"/>          <!-- Nombre de la tarea -->
      <field name="prioridad"/>      <!-- Nivel de prioridad -->
      <field name="urgente"/>        <!-- Calculado automáticamente -->
      <field name="realizada"/>      <!-- Estado de finalización -->
      <field name="fecha_inicio"/>   <!-- Fecha de inicio -->
      <field name="fecha_fin"/>      <!-- Fecha de fin -->
    </list>
  </field>
</record>
```

Figura 11: Actualizamos el bloque de la vista list para mostrar los nuevos campos de fechas de inicio y fin de la tarea.

En la vista de lista, agregamos los nuevos campos de fecha de inicio y fecha de final de la tarea como nuevas columnas.

```xml
<!-- ================================================
     VISTA DE KANBAN
     ================================================ -->
<record id="view_lista_tareas_kanban" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.kanban</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo kanban -->
    <kanban default_order_by="realizada">
      <field name="tarea"/>
      <field name="prioridad"/>
      <field name="urgente"/>
      <field name="realizada"/>

      <templates>
        <!-- Tarjeta de las tareas de kanban -->
        <t t-name="kanban-box">
          <div class="oe_kanban_card oe_kanban_global_click">
            <strong><field name="tarea"/></strong>
            <div>
              <div>Prioridad: <field name="prioridad"/></div>
              <div>Urgente: <field name="urgente"/></div>
              <div>Realizada: <field name="realizada"/></div>
              <div>Fecha de inicio: <field name="fecha_inicio"/></div>
              <div>Fecha de fin: <field name="fecha_fin"/></div>
            </div>
          </div>
        </t>
      </templates>
    </kanban>
  </field>
</record>
```

Figura 12: Actualizamos el bloque de la vista kanban para mostrar los nuevos campos de fechas de inicio y fin de la tarea.

En la vista de kanban, agregamos los nuevos campos de fecha de inicio y fecha de final de la tarea en la tarjeta del kanban.

## 2.5. Comprobación de la vista Calendar

A continuación vamos a enseñar a través de capturas de pantalla el correcto funcionamiento de la vista calendar que acabamos de implementar.

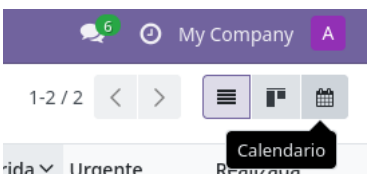



Figura 13: Como se pueder ver en esta captura para poder cambiar de vista nos iremos a estos iconos y pulsaremos en el que pone calendario.

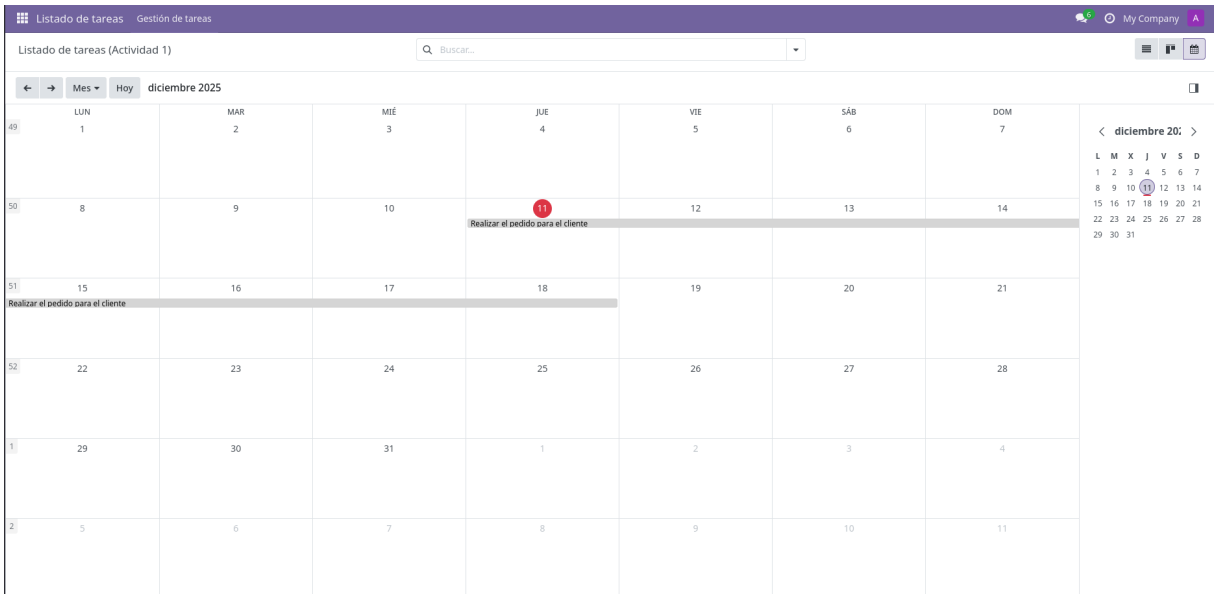



Figura 14: En esta captura podemos ver la vista modo calendar de nuestro módulo de listas de tareas.

Aquí podemos ver como la tarea que creamos de **realizar un pedido para un cliente**, empieza el jueves 4 y termina el viernes 18 de diciembre de 2025, lo que muestra que tenemos la vista del calendario bien implementada.

# 3. Actividad 02 - Biblioteca de cómics

## 3.1. Objetivo

Para esta actividad se nos pide ampliar el módulo del ejemplo 3 de la biblioteca de cómics simple para agregar la gestión de socios y la gestión de ejemplares de préstamo. Para ello crearemos nuevos modelos que nos permiten almacenar los datos de los socios y de los ejemplares, así como poder registrar los préstamos indicando el socio, la fecha de inicio y la fecha de devolución, aplicando las restricciones indicadas a las fechas.

## 3.2. Modelo Socios

A continuación vamos a crear el modelo de socios para nuesto módulo, para ello tendremos que crear la vista y el modelo además de realizar unas modificaciones para emplearlo.

```python
# -*- coding: utf-8 -*-
from odoo import models, fields

# Definimos el modelo Biblioteca socio
class BibliotecaSocio(models.Model):
    # Nombre del modelo
    _name = "biblioteca.socio"
    # Descripcion del modelo
    _description = "Socio de la bilioteca"
    # Usamos el id_socio como nobre del socio
    _rec_name = "id_socio"

    # Atributos del modelo: identificador, nombre y apellido
    id_socio = fields.Char("ID socio", required=True, index=True)
    nombre = fields.Char("Nombre", required=True)
    apellido = fields.Char("Apellido", required=True)

    #Constraints de SQL del modelo: el identificador deber ser único
    _sql_constraints = [
        ("identificador_uniq", "UNIQUE(id_socio)", "El identificador del socio debe ser único.")
    ]
```

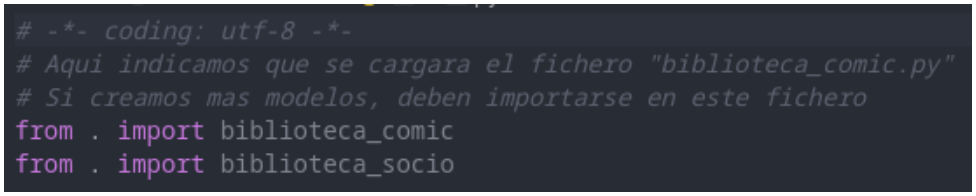
Figura 15: Captura del archivo `biblioteca_socio.py` donde tenemos el modelo para gestionar socios.

Creamos el archivo `biblioteca_socio.py` y definimos el modelo "biblioteca.socio", que se encargará de gestionar los socios de la biblioteca de cómics. Cada socio tiene un identificador único, además de su nombre y apellido.

Los atributos principales del modelo son:

- id_socio: identificador único para cada socio.
- nombre: nombre del socio.
- apellido: apellido del socio.

Además, usamos una restricción SQL para asegurar que el `id_socio` sea único.



```python
# -*- coding: utf-8 -*-
# Aqui indicamos que se cargara el fichero "biblioteca_comic.py"
# Si creamos mas modelos, deben importarse en este fichero
from . import biblioteca_comic
from . import biblioteca_socio
```

Figura 16: Captura de que importamos el nuevo modelo socio en `models/__init__.py`.

Para que Odoo cargue nuestro modelo socios al iniciar el módulo, importamos el modelo en el archivo `models/__init__.py` como ya teniamos el del modelo de cómic.

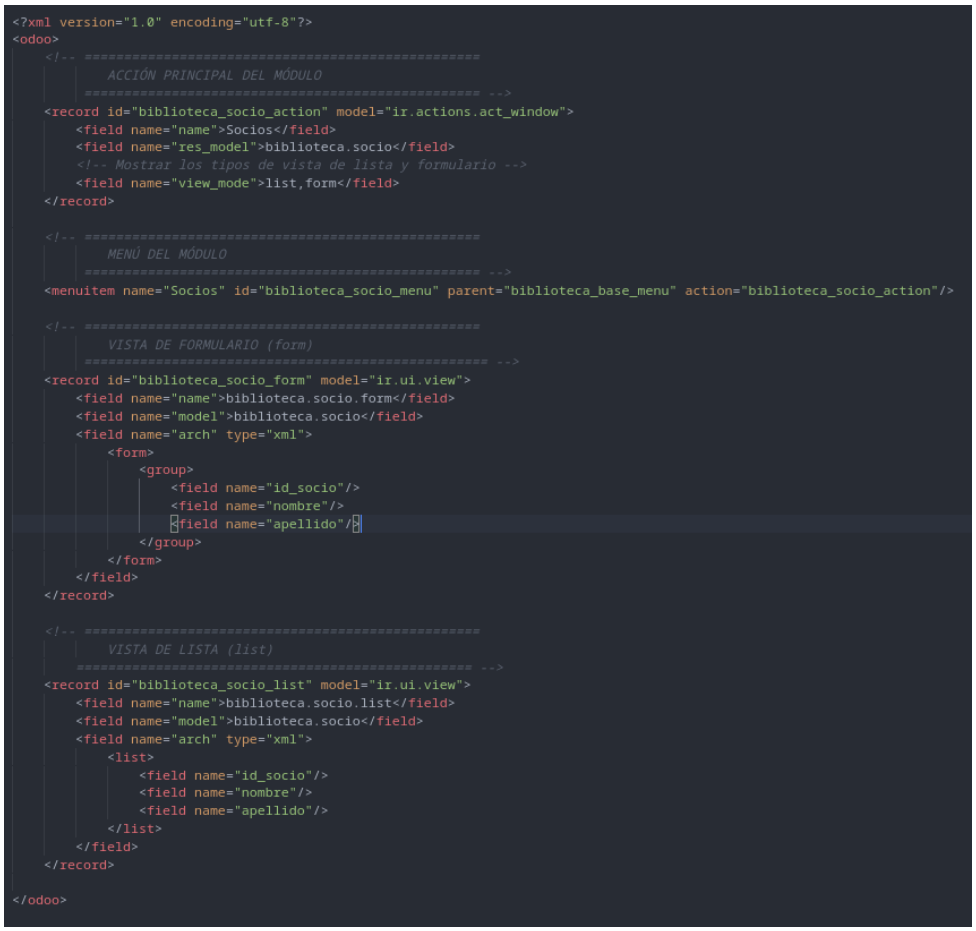



Figura 17: Captura del archivo `biblioteca_socio.xml` donde tenemos definida la vista para socios.

Creamos el archivo `biblioteca_socio.xml` donde creamos la vista `socio` para poder gestionar los socios desde una interfaz donde tenemos lo siguiente:

- **Vista formulario:** Esto nos permite tener un formulario donde podremos crear o editar la información de los socios.
- **Vista lista:** Esto nos muestra todos los socios en un listado formato tabla con los campos: identificador, nombre y apellido.

También agregamos la acción y el menú correspondientes para poder acceder a la vista de socios desde la interfaz.

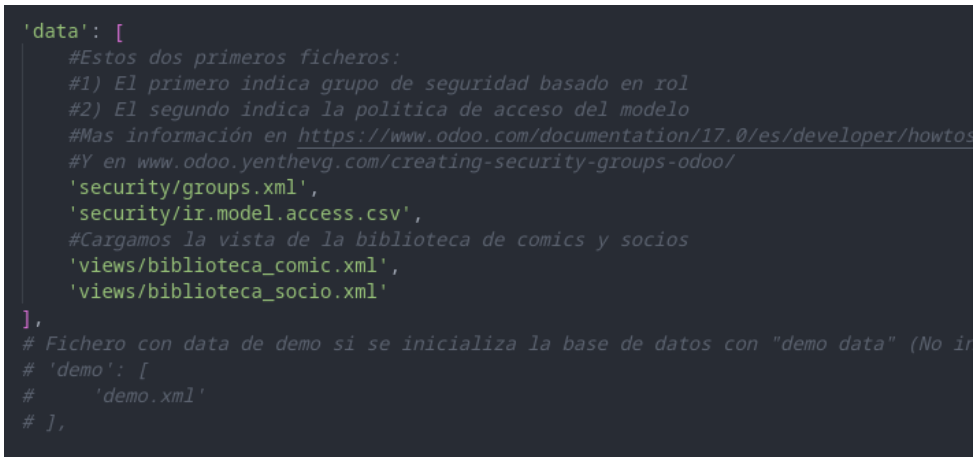



Figura 18: Captura de que incluimos la nueva vista de socio en `__manifest__.py`.

Para que Odoo cargue nuestro vista de socios, tenemos que agregarlo dentro de `data` en el `__manifest__.py` justo debajo de la vista de cómics.

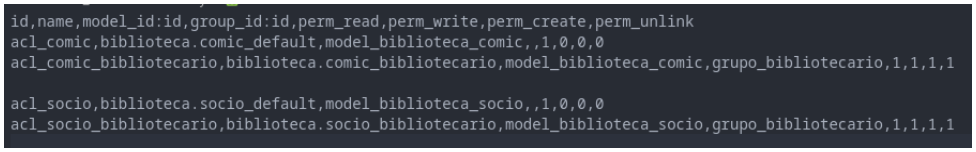
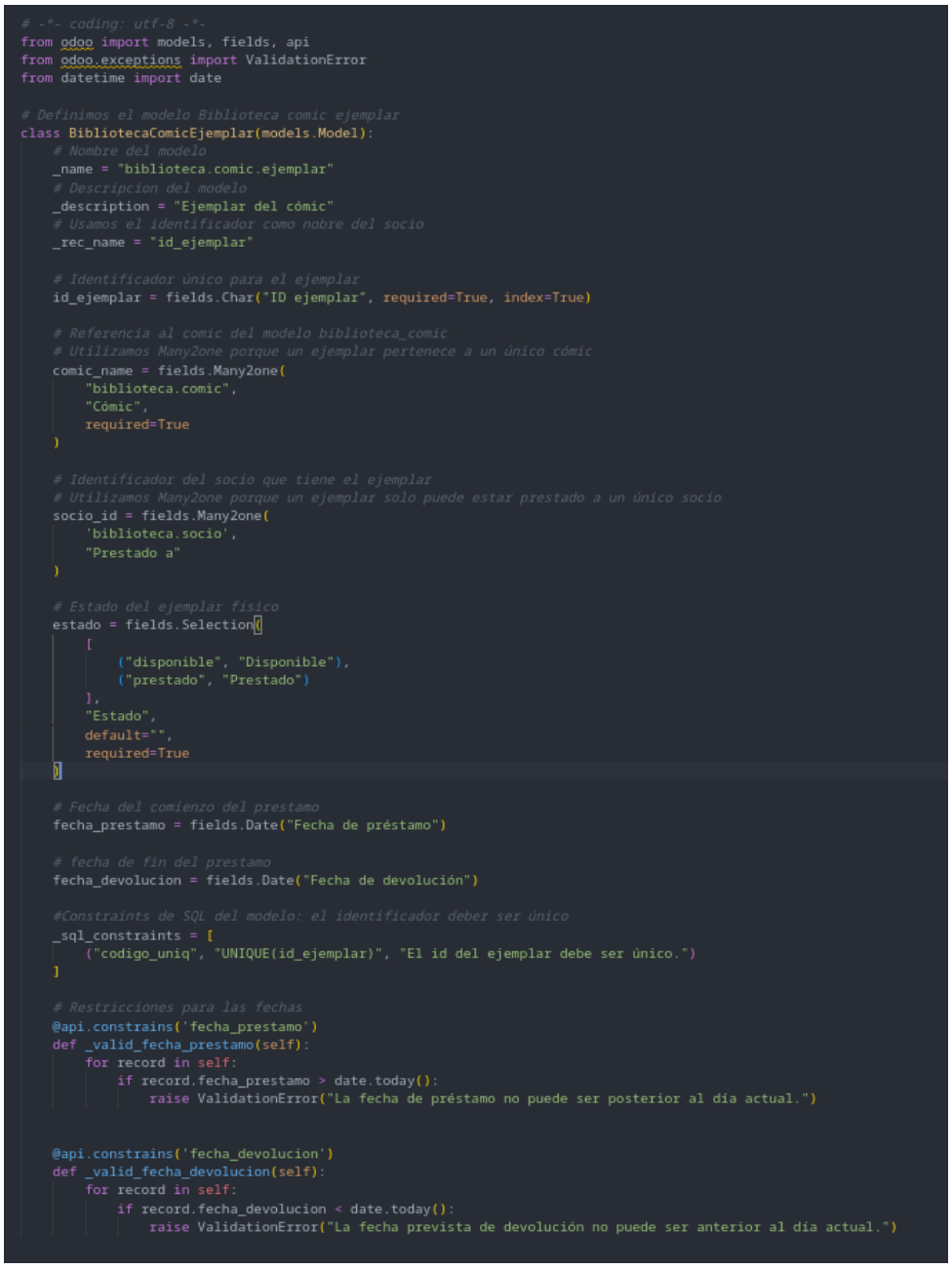



Figura 19: Captura donde actualizamos las reglas de acceso donde asignamos los permisos para socios.

Ahora, definimos las reglas de acceso, permitiendo a todos los usuarios poder ver los socios, pero solo los miembros del **grupo bibliotecario** podrán crear, editar y eliminar los socios.

## 3.3. Modelo Ejemplares

A continuación vamos a crear el modelo de ejemplares para nuesto módulo, para ello tendremos que crear la vista y el modelo además de realizar unas modificaciones para poder emplearlo.



```python
# -*- coding: utf-8 -*-
from odoo import models, fields, api
from odoo.exceptions import ValidationError
from datetime import date

# Definimos el modelo Biblioteca comic ejemplar
class BibliotecaComicEjemplar(models.Model):
    # Nombre del modelo
    _name = "biblioteca.comic.ejemplar"
    # Descripcion del modelo
    _description = "Ejemplar del cómic"
    # Usamos el identificador como nobre del socio
    _rec_name = "id_ejemplar"

    # Identificador único para el ejemplar
    id_ejemplar = fields.Char("ID ejemplar", required=True, index=True)

    # Referencia al comic del modelo biblioteca_comic
    # Utilizamos Many2one porque un ejemplar pertenece a un único cómic
    comic_name = fields.Many2one(
        "biblioteca.comic",
        "Cómic",
        required=True
    )

    # Identificador del socio que tiene el ejemplar
    # Utilizamos Many2one porque un ejemplar solo puede estar prestado a un único socio
    socio_id = fields.Many2one(
        'biblioteca.socio',
        "Prestado a"
    )

    # Estado del ejemplar físico
    estado = fields.Selection(
        [
            ("disponible", "Disponible"),
            ("prestado", "Prestado")
        ],
        "Estado",
        default="",
        required=True
    )

    # Fecha del comienzo del prestamo
    fecha_prestamo = fields.Date("Fecha de préstamo")

    # fecha de fin del prestamo
    fecha_devolucion = fields.Date("Fecha de devolución")

    #Constraints de SQL del modelo: el identificador deber ser único
    _sql_constraints = [
        ("codigo_uniq", "UNIQUE(id_ejemplar)", "El id del ejemplar debe ser único.")
    ]

    # Restricciones para las fechas
    @api.constrains('fecha_prestamo')
    def _valid_fecha_prestamo(self):
        for record in self:
            if record.fecha_prestamo > date.today():
                raise ValidationError("La fecha de préstamo no puede ser posterior al día actual.")


    @api.constrains('fecha_devolucion')
    def _valid_fecha_devolucion(self):
        for record in self:
            if record.fecha_devolucion < date.today():
                raise ValidationError("La fecha prevista de devolución no puede ser anterior al día actual.")
```

Figura 20: Captura del archivo `biblioteca_ejemplares.py` donde tenemos el modelo para gestionar los ejemplares.

Creamos el archivo `biblioteca_ejemplar.py` y definimos el modelo `"biblioteca.comic.ejemplar"`. Este modelo nos permite gestionar cada ejemplar de un comic de nuestra biblioteca. Cada ejemplar tiene un identificador único, una referencia al cómic al que pertenece, un estado y opcionalmente, un socio al que esta prestado y las fechas de prestamo y devolución.

Los atributos principales del modelo son:

- id_ejemplar: identificador único para cada ejemplar.
- comic_name: tenemos una relación `Many2one` hacia el modelo de cómic, ya que cada ejemplar pertenece a un único cómic.

- socio_id: tenemos una relación `Many2one` hacia el modelo de socio, ya que cada ejemplar esta prestado a un socio.
- estado: indica si el ejemplar está disponible o está prestado.
- fecha_prestamo: fecha en la que se presto el ejemplar.
- fecha_devolucion: fecha prevista en la que se devolvera el ejemplar.

Además, usamos una restricción SQL para asegurar que el `id_ejemplar` sea único.

Agregamos las restricciones para las fechas:

Incluimos dos comprobaciones mediante `@api.constrains`, así si se intenta guardar un ejemplar que incumpla alguna de estás reglas que nos salte un error:

1. Que la fecha de préstamo no puede ser posterior al día actual.
2. Que la fecha de devolución no puede ser anterior al día actual.

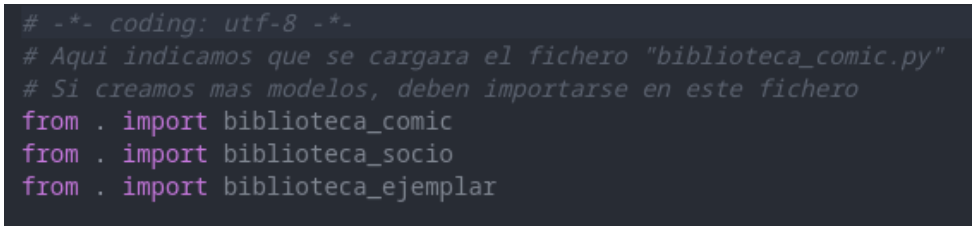



Figura 21: Captura de que importamos el nuevo modelo ejemplar en `models/__init__.py`.

Para que Odoo cargue nuestro modelo ejemplar al iniciar el módulo, importamos el modelo en el archivo `models/__init__.py` como ya teniamos el del modelo de cómic y socio.

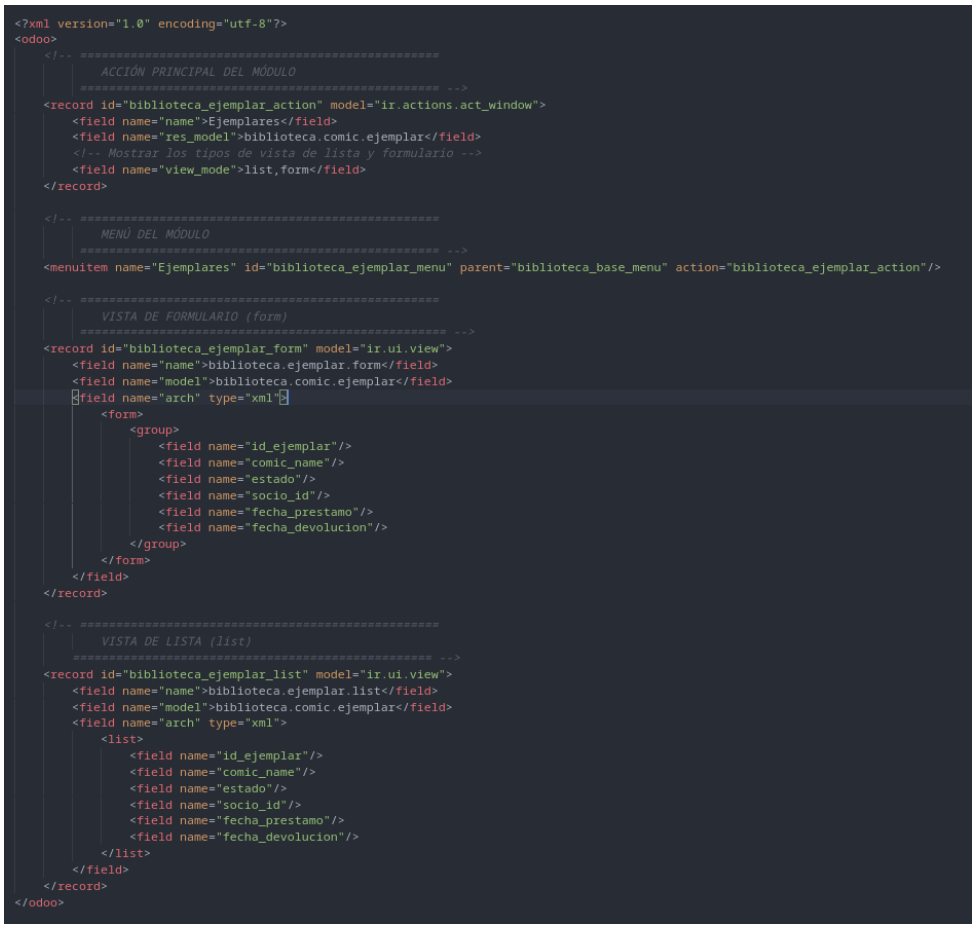



Figura 22: Captura del archivo `biblioteca_ejemplar.xml` donde tenemos definida la vista para los ejemplares.

Creamos el archivo `biblioteca_ejemplar.xml` donde creamos la vista `ejemplar` para poder gestionar los ejemplares desde una interfaz donde tenemos lo siguiente:

- **Vista formulario:** Esto nos permite tener un formulario donde podremos crear o editar la información de los ejemplares.
- **Vista lista:** Esto nos muestra todos los ejemplares en un listado formato tabla con los campos: identificador del ejemplar, nombre del cómic, estado, identificador del socio, fecha del prestamo y de devolución.

También agregamos la acción y el menú correspondientes para poder acceder a la vista de ejemplares desde la interfaz.

```
'data': [
    #Estos dos primeros ficheros:
    #1) El primero indica grupo de seguridad basado en rol
    #2) El segundo indica la politica de acceso del modelo
    #Mas información en https://www.odoo.com/documentation/17.0/es/developer/howtos/rdtraining/05_securityintro.html
    #Y en www.odoo.yenthevg.com/creating-security-groups-odoo/
    'security/groups.xml',
    'security/ir.model.access.csv',
    #Cargamos la vista de la biblioteca de comics y socios
    'views/biblioteca_comic.xml',
    'views/biblioteca_socio.xml',
    'views/biblioteca_ejemplar.xml'
],
# Fichero con data de demo si se inicializa la base de datos con "demo data" (No incluido en ejemplo)
# 'demo': [
#     'demo.xml'
# ],
```

Figura 23: Captura de que incluimos la nueva vista de ejemplar en `__manifest__.py`.

Para que Odoo cargue nuestro vista de ejemplares, tenemos que agregarlo dentro de `data` en el `__manifest__.py` justo debajo de la vista de socios.

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
acl_comic,biblioteca.comic_default,model_biblioteca_comic,,1,0,0,0
acl_comic_bibliotecario,biblioteca.comic_bibliotecario,model_biblioteca_comic,grupo_bibliotecario,1,1,1,1

acl_socio,biblioteca.socio_default,model_biblioteca_socio,,1,0,0,0
acl_socio_bibliotecario,biblioteca.socio_bibliotecario,model_biblioteca_socio,grupo_bibliotecario,1,1,1,1

acl_ejemplar,biblioteca.ejemplar_default,model_biblioteca_comic_ejemplar,,1,0,0,0
acl_ejemplar_bibliotecario,biblioteca.ejemplar_bibliotecario,model_biblioteca_comic_ejemplar,grupo_bibliotecario,1,1,1,1
```

Figura 24: Captura donde actualizamos las reglas de acceso donde asignamos los permisos para ejemplares.

Ahora, definimos las reglas de acceso, permitiendo a todos los usuarios poder ver los ejemplares, pero solo los miembros del **grupo bibliotecario** podrán crear, editar y eliminar los ejemplares.

## 3.4. Comprobación de su funcionamiento

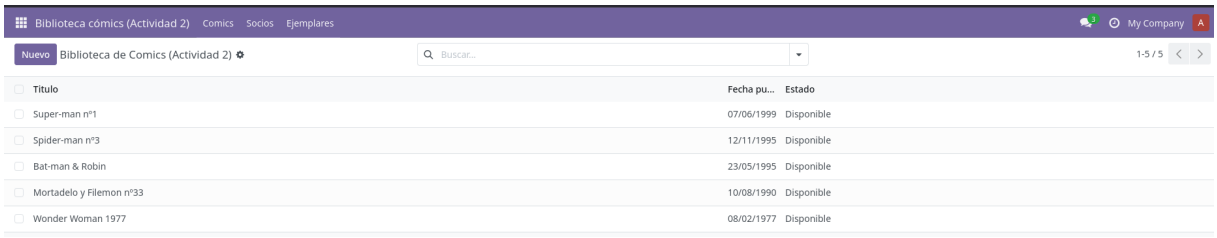



Figura 25: En esta captura tenemos la lista con todos los cómics que creamos para nuestra biblioteca.

Como podemos ver tenemos creados cinco cómics y vemos que se muestra correctamente en el formato lista, ademas podemos ver en el menú las opciones cómics, socios y ejemplares, para movernos de modelo.

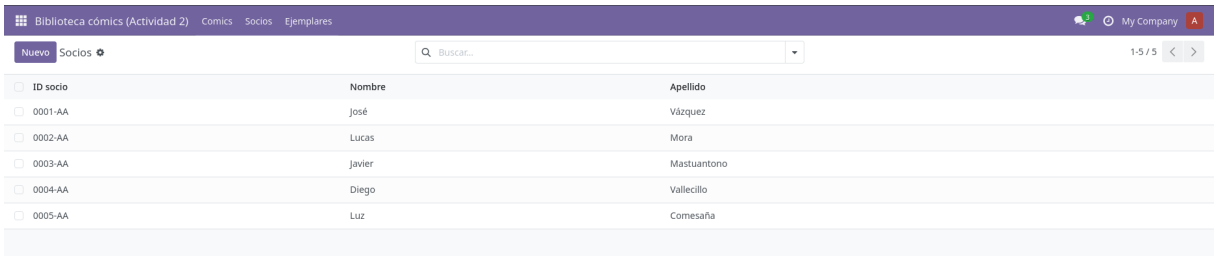



Figura 26: En esta captura tenemos la lista con todos los socios que hemos creamos para nuestra biblioteca.

Como podemos ver estamos en la parte del modulo de socios donde los gestionamos, creamos cinco socios de ejemplos y los mostramos en el formato lista correctamente.

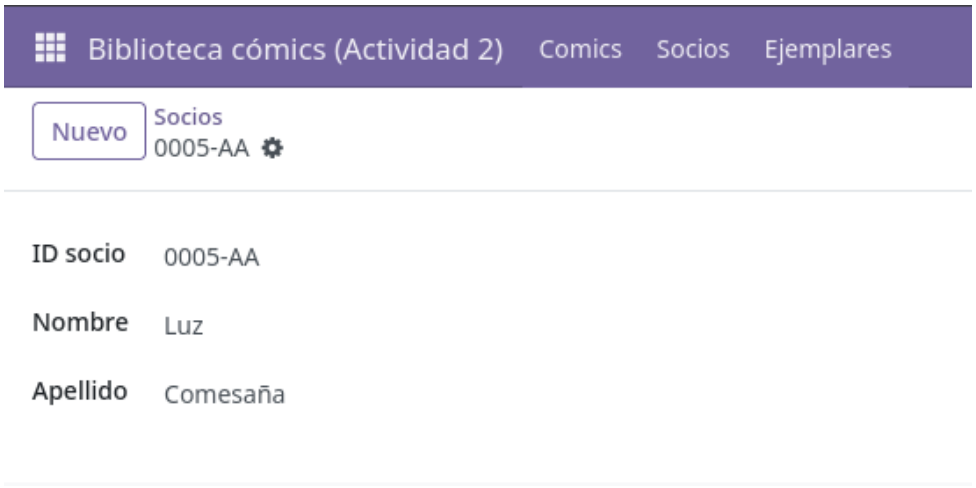



Figura 27: En está captura tenemos el formulario para crear y editar los socios.

Como podemos ver ahora estamos en la pantalla de crear o editar un socio donde podemos ver que tenemos los campos: identificador, nombre y apellido.

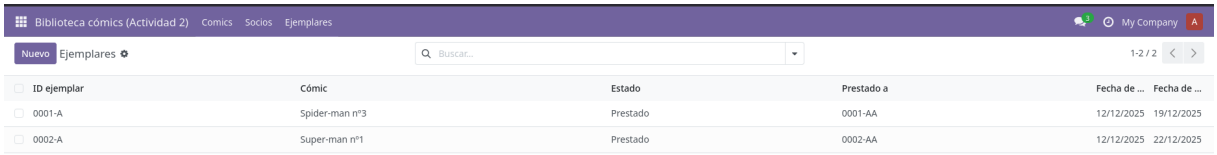



Figura 28: En esta captura tenemos la lista con todos los ejemplares que hemos creamos para nuestra biblioteca.

Ahora nos encontramos en la parte del modulo de ejemplares donde los gestionamos, he creado 2 ejemplares de ejemplos y los mostramos en el formato lista correctamente.


Figura 29: En está captura tenemos el formulario para crear y editar nuestros ejemplares.

Como podemos ver ahora estamos en la pantalla de crear o editar un ejemplar donde podemos ver que tenemos los campos: identificador del ejemplar, nombre del cómic, estado, a que socio está prestado, en que fecha se presto y cuando se va a devolver.

Por último vamos a mostrar las comprobaciones de que nuestras restricciones en las fechas en ejemplares están funcionando correctamente.

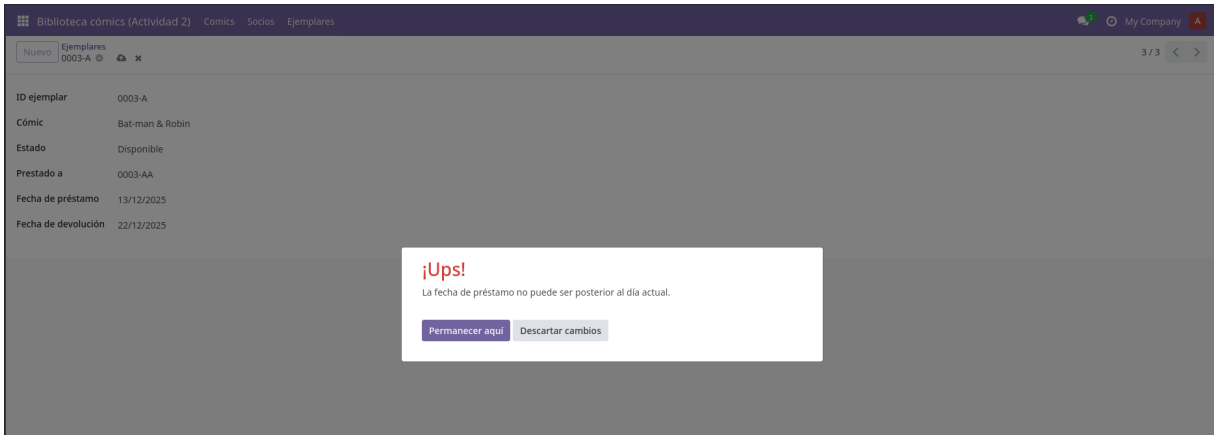



Figura 30: En está captura comprobamos que la fecha de préstamo no puede ser posterior al día actual.

Aquí podemos ver como al intentar guardar un nuevo ejemplar con la fecha de prestamo 13/12/2025 nos salta el error ya que el dia en la intentamos registrar el ejemplar era 12/12/2025 y como definimos en la restricción no podemos registrar un inicio de prestamo en un fecha futura.

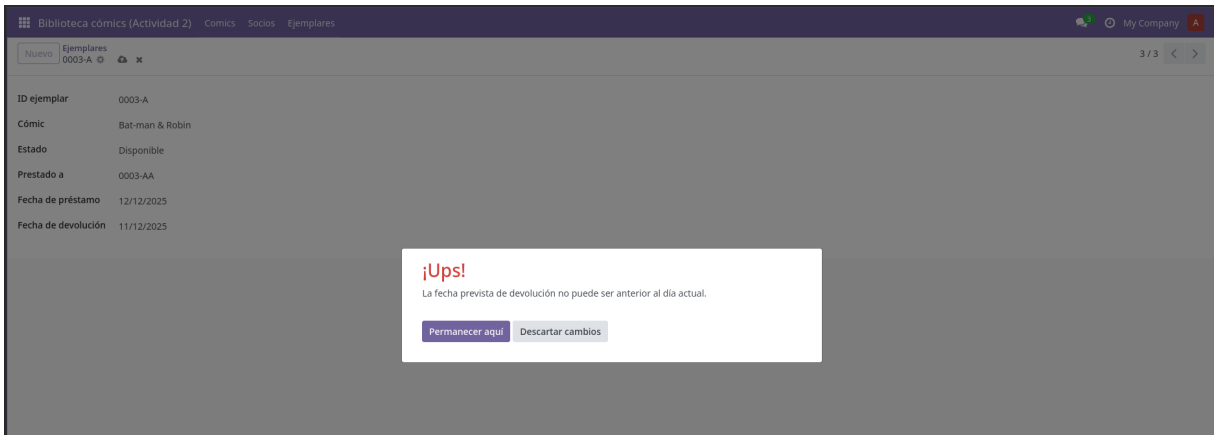



Figura 31: En está captura comprobamos que la fecha prevista de devolución no puede ser anterior al día actual.

Aquí podemos ver como al intentar guardar un nuevo ejemplar con la fecha de devolución 11/12/2025 nos salta el error ya que el dia en la intentamos registrar el ejemplar era 12/12/2025 y como definimos en la restricción no podemos registrar una devolución de prestamo en un fecha ya pasada al día actual.

# 4. Actividad 03 - Pacientes y médicos

## 4.1. Objetivo

Para esta actividad tendremos que crear un módulo para gestionar los pacientes y médicos de un hospital, teniendo también un registro de las consultas. Para ello deberemos de crear los modelos para almacenar los datos de los médicos, pacientes y de las consultas, donde registramos cada vez que un médico atiente a un paciente incluyendo el diagnostico dado por el médico.

RECORDATORIO:

Para poder crear la estructura básica de un módulo, usaremos el comando **scaffold** a continuación mostraré los pasos a seguir:

Comando para entrar en nuestro contenedor de docker

```
docker exec -it <CONTAINER> bash
```

Comando para crear la estructura de nuestro módulo

```
odoo scaffold <MODULO_NAME> /mnt/extra-addons
```

## 4.2. Modelo Paciente

A continuación vamos a crear el modelo de pacientes donde almacenaremos los datos de los pacientes, para ello tendremos que crear la vista y su modelo además de referenciar el nuevo modelo en el `__init__.py` de models y la vista en nuestro `__manifest__.py`.

```python
# -*- coding: utf-8 -*-
from odoo import models, fields, api

# Definimos el modelo del paciente
class Paciente(models.Model):
    # Nombre del modelo
    _name = "paciente"
    # Descripcion del modelo
    _description = "Paciente de un hospital"
    # Usamos el nombre completo como nombre visible del registro
    _rec_name = "nombre_completo"

    # Atributos del modelo: identificador, nombre, apellido, sintomas y y obtenemos el nombre completo
    id_paciente = fields.Char("ID paciente", required=True, index=True)
    nombre = fields.Char("Nombre", required=True)
    apellidos = fields.Char("Apellidos", required=True)
    sintomas = fields.Text('Síntomas', required=True)
    nombre_completo = fields.Char("Paciente", compute="_compute_nombre_completo", store=True)

    @api.depends('nombre', 'apellidos')
    def _compute_nombre_completo(self):
        for record in self:
            record.nombre_completo = f"{record.nombre} {record.apellidos}"

    #Constraints de SQL del modelo: el identificador deber ser único
    _sql_constraints = [
        ("identificador_uniq", "UNIQUE(id_paciente)", "El identificador del paciente debe ser único.")
    ]
```

Figura 32: Captura del archivo `paciente.py` donde tenemos el modelo para gestionar a los pacientes del hospital.

Creamos el archivo `paciente.py` y definimos el modelo `"paciente"`. Este modelo nos permite gestionar los pacientes del hospital. Cada paciente cuenta con un un identificador único, sus datos personales y los síntomas que padece. Además contamos con un helper para poder recuperar su nombre completo y facilitar la visualización.

Los atributos principales del modelo son:

- id_paciente: identificador único para cada paciente.
- nombre: nombre del paciente.
- apellidos: apellidos del paciente.
- sintomas: descripción de los síntomas del paciente.
- nombre_completo: combina el nombre + apellidos para obtener el nombre completo.

Además, usamos una restricción SQL para asegurar que el `id_paciente` sea único.

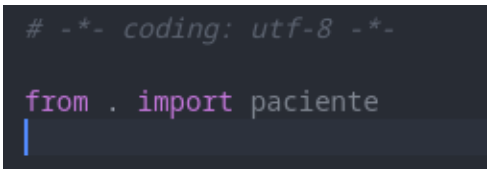



Figura 33: Captura de que importamos el nuevo modelo paciente en `models/__init__.py`.

Para que Odoo cargue nuestro modelo paciente al iniciar el módulo, importamos el modelo en el archivo `models/__init__.py`.

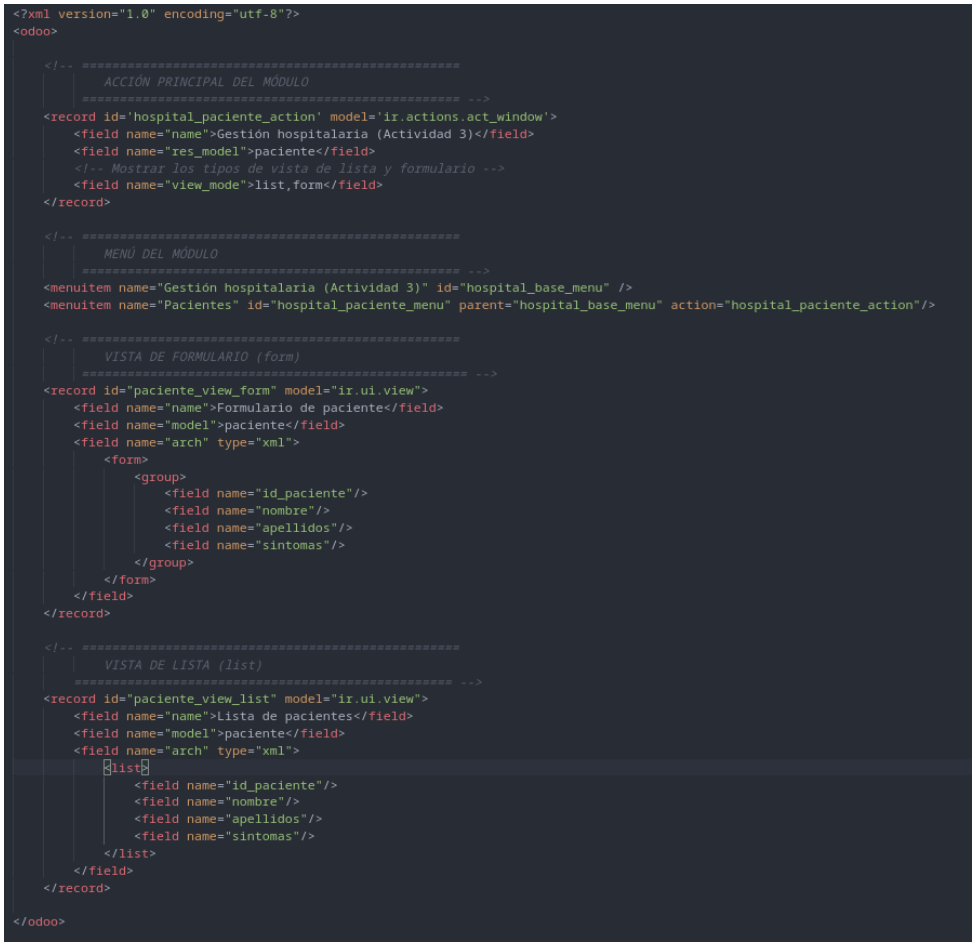



Figura 34: Captura del archivo `paciente.xml` donde tenemos definida la vista para los pacientes del hospital.

Creamos el archivo `paciente.xml` donde creamos la vista `paciente` para poder gestionar los pacientes desde una interfaz donde tenemos lo siguiente:

- **Vista formulario:** Esto nos permite tener un formulario donde podremos crear o editar la información de los pacientes.
- **Vista lista:** Esto nos muestra todos los pacientes en un listado formato tabla con los campos: identificador del paciente, nombre, apellidos y síntomas.

También agregamos la acción y el menú correspondientes para poder acceder a la vista de pacientes desde la interfaz.


Figura 35: Captura de que incluimos la nueva vista de paciente en `__manifest__.py`.

Para que Odoo cargue nuestro vista de pacientes, tenemos que agregarlo dentro de `data` en el `__manifest__.py`.

## 4.3. Modelo Médico

A continuación vamos a crear el modelo de médicos donde almacenaremos los datos de los médicos, para ello tendremos que crear la vista y su modelo además de referenciar el nuevo modelo en el `__init__.py` de models y la vista en nuestro `__manifest__.py`.

```python
# -*- coding: utf-8 -*-
from odoo import models, fields, api

# Definimos el modelo del paciente
class Paciente(models.Model):
    # Nombre del modelo
    _name = "paciente"
    # Descripcion del modelo
    _description = "Paciente de un hospital"
    # Usamos el nombre completo como nombre visible del registro
    _rec_name = "nombre_completo"

    # Atributos del modelo: identificador, nombre, apellido, sintomas y y obtenemos el nombre completo
    id_paciente = fields.Char("ID paciente", required=True, index=True)
    nombre = fields.Char("Nombre", required=True)
    apellidos = fields.Char("Apellidos", required=True)
    sintomas = fields.Text('Síntomas', required=True)
    nombre_completo = fields.Char("Paciente", compute="_compute_nombre_completo", store=True)

    @api.depends('nombre', 'apellidos')
    def _compute_nombre_completo(self):
        for record in self:
            record.nombre_completo = f"{record.nombre} {record.apellidos}"

    #Constraints de SQL del modelo: el identificador deber ser único
    _sql_constraints = [
        ("identificador_uniq", "UNIQUE(id_paciente)", "El identificador del paciente debe ser único.")
    ]
```

Figura 36: Captura del archivo `medico.py` donde tenemos el modelo para gestionar a los médicos del hospital.

Creamos el archivo `medico.py` y definimos el modelo `"medico"`. Este modelo nos permite almacenar la información de los médicos del hospital. Cada médico cuenta con un un número de colegiado único, además de su nombre y apellidos. Y como hacemos en el modelo de paciente, contamos con un helper para poder recuperar su nombre completo y facilitar la visualización.

Los atributos principales del modelo son:

- num_colegiado: número de colegiado del médico, que se usa como su identificador.
- nombre: nombre del médico.
- apellidos: apellidos del médico.
- nombre_completo: combina el nombre + apellidos para obtener el nombre completo.

Además, usamos una restricción SQL para asegurar que el `num_colegiado` sea único.

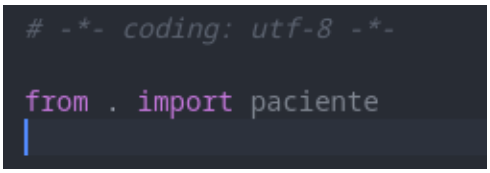



Figura 37: Captura de que importamos el nuevo modelo médico en `models/__init__.py`.

Para que Odoo cargue nuestro modelo médico al iniciar el módulo, importamos el modelo en el archivo `models/__init__.py` como ya teniamos el de paciente.

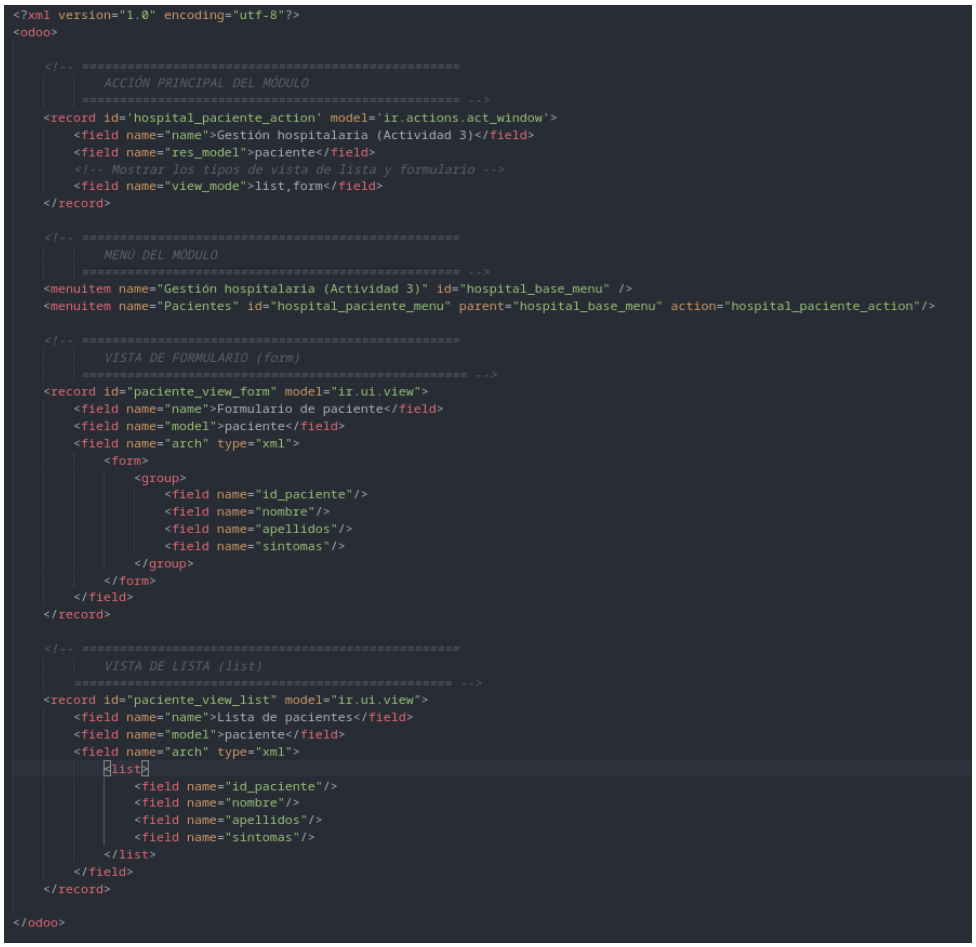



Figura 38: Captura del archivo `medico.xml` donde tenemos definida la vista para los médicos del hospital.

Creamos el archivo `medico.xml` donde creamos la vista `medico` para poder gestionar los médicos desde una interfaz donde tenemos lo siguiente:

- **Vista formulario:** Esto nos permite tener un formulario donde podremos crear o editar la información de los médicos.
- **Vista kanban:** Esto nos muestra a los médicos en formato de tarjetas, facilitando la visualización de estos, y mostramos su número de colegiado, nombre y apellidos.

También agregamos la acción y el menú correspondientes para poder acceder a la vista de médicos desde la interfaz.


Figura 39: Captura de que incluimos la nueva vista de médico en `__manifest__.py`.

Para que Odoo cargue nuestro vista de médicos, tenemos que agregarlo dentro de `data` en el `__manifest__.py` justo debajo de la vista de paciente.

## 4.4. Modelo Consulta

A continuación vamos a crear el modelo para las consultas que registrará cada vez que un médico atiende a un paciente, para ello tendremos que crear la vista y su modelo además de referenciar el nuevo modelo en el `__init__.py` de models y la vista en nuestro `__manifest__.py`.

```python
# -*- coding: utf-8 -*-
from odoo import models, fields, api

# Definimos el modelo de consulta
class Consulta(models.Model):
    # Nombre del modelo
    _name = "consulta"
    # Descripcion del modelo
    _description = "Registro de las consultas"
    # Usamos el id_consulta como nombre visible del registro
    _rec_name = "id_consulta"

    # Atributos del modelo: identificador, medico, paciente, diagnostico y fecha de la cita.
    id_consulta = fields.Char("ID consulta", required=True, index=True)
    medico_id = fields.Many2one(
        "medico",
        "medico",
        required=True
    )
    paciente_id = fields.Many2one(
        "paciente",
        "paciente",
        required=True
    )
    diagnostico = fields.Text('Diagnóstico', required=True)
    fecha_cita = fields.Date("Fecha de la cita", required=True)


    #Constraints de SQL del modelo: el identificador deber ser único
    _sql_constraints = [
        ("identificador_uniq", "UNIQUE(id_consulta)", "El identificador de la consulta debe ser único.")
    ]
```

Figura 40: Captura del archivo `consulta.py` donde tenemos el modelo para gestionar a las cosultas médicas.

Creamos el archivo `consulta.py` y definimos el modelo `"consulta"`. Este modelo nos permite almacenar la información de cada consulta realizada. Cada consulta tiene un identificador único, y relaciona un médico con un paciente, registrando además el diagnóstico y la fecha de la cita.

Los atributos principales del modelo son:

- id_consulta: identificador único de la consulta.
- medico_id: indica que médico hace la consulta, y tenemos una relación `Many2one`, ya que cada médico puede atender varios pacientes.
- paciente_id: indica el paciente atendido, y tenemos una relación `Many2one`, ya que cada paciente puede haber sido atendido por varios médicos.
- diagnostico: diagnóstico emitido por el médico.
- fecha_cita: fecha en la que se realiza la consulta médica.

Además, usamos una restricción SQL para asegurar que el `id_consulta` sea único.

```python
# -*- coding: utf-8 -*-

from . import paciente
from . import medico
from . import consulta
```

Figura 41: Captura de que importamos el nuevo modelo consulta en `models/__init__.py`.

Para que Odoo cargue nuestro modelo consulta al iniciar el módulo, importamos el modelo en el archivo `models/__init__.py` como ya teniamos el del modelo de paciente y médico.

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <!-- ================================================
         ACCIÓN PRINCIPAL DEL MÓDULO
         ================================================ -->
    <record id="hospital_consulta_action" model="ir.actions.act_window">
        <field name="name">Consultas</field>
        <field name="res_model">consulta</field>
        <!-- Mostrar los tipos de vista de lista, calendario y formulario -->
        <field name="view_mode">list,calendar,form</field>
    </record>

    <!-- ================================================
         MENÚ DEL MÓDULO
         ================================================ -->
    <menuitem name="Consultas" id="hospital_consulta_menu" parent="hospital_base_menu" action="hospital_consulta_action"/>

    <!-- ================================================
         VISTA DE FORMULARIO (form)
         ================================================ -->
    <record id="consulta_view_form" model="ir.ui.view">
       <field name="name">Formulario de consultas</field>
        <field name="model">consulta</field>
        <field name="arch" type="xml">
            <form>
                <group>
                    <field name="id_consulta"/>
                    <field name="medico_id"/>
                    <field name="paciente_id"/>
                    <field name="diagnostico"/>
                    <field name="fecha_cita"/>
                </group>
            </form>
        </field>
    </record>

    <!-- ================================================
         VISTA DE LISTA (list)
         ================================================ -->
    <record id="consulta_view_list" model="ir.ui.view">
        <field name="name">Lista de las consultas</field>
        <field name="model">consulta</field>
        <field name="arch" type="xml">
            <list>
                <field name="id_consulta"/>
                <field name="medico_id"/>
                <field name="paciente_id"/>
                <field name="diagnostico"/>
                <field name="fecha_cita"/>
            </list>
        </field>
    </record>

    <!-- ================================================
         VISTA DE CALENDARIO
         ================================================ -->
    <record id="consulta_view_calendar" model="ir.ui.view">
      <field name="name">Consultas en calendario</field>
      <field name="model">consulta</field>
      <field name="arch" type="xml">
        <!-- Vista tipo calendar -->
        <calendar date_start="fecha_cita" date_stop="fecha_cita">
          <field name="id_consulta"/>
          <field name="medico_id"/>
          <field name="paciente_id"/>
          <field name="diagnostico"/>
        </calendar>
      </field>
    </record>

</odoo>
```

Figura 42: Captura del archivo `consulta.xml` donde tenemos definida la vista para las consultas médicas.

Creamos el archivo `consulta.xml` donde creamos la vista `consulta` para poder gestionar las consultas desde una interfaz donde tenemos lo siguiente:

- **Vista formulario:** Esto nos permite tener un formulario donde podremos crear o editar la información de las consultas.
- **Vista lista:** Esto nos muestra todas las consultas en un listado formato tabla con los campos: identificador de la consulta, médico que atendio, paciente atendido, diagnóstico y fecha de la cita.
- **Vista calendario:** Esto nos muestra todas las consultas en un calendario en la fecha en la que fue y nos permite visualizar mejor las citas médicas porque es como habitualmente estamos acostumbrados a verlas.

También agregamos la acción y el menú correspondientes para poder acceder a la vista de consultas desde la interfaz.


Figura 43: Captura de que incluimos la nueva vista de consulta en `__manifest__.py`.

Para que Odoo cargue nuestro vista de consulta, tenemos que agregarlo dentro de `data` en el `__manifest__.py` justo debajo de la vista de paciente y medico.

## 4.5. Comprobación de su funcionamiento

Antes de mostrar el funcionamiento vamos a mostrar el error que nos sucedio antes de poder probarlo.

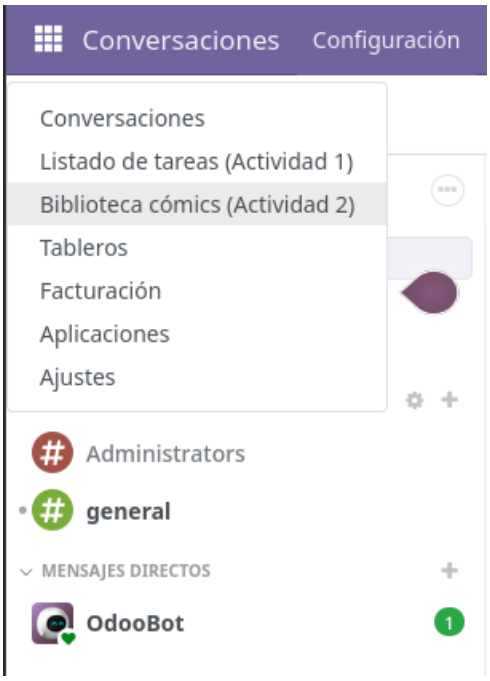



Figura 44: Captura de que no nos mostraba nuestro módulo **Gestión hospitalaria (Actividad 3)**.

Después de haber instalado nuestro módulo **gestión hospitalaria** no dirigimos al menú para entrar al módulo y no nos salía en el menú, esto nos pasaba porque no teníamos permisos, en el módulo anterior no tuvimos es problema porque al tener el permiso del modelo comics lo replicamos para ejemplares y socios. Pero en este modulo no los pusimos y por eso no se nos mostraba. A continuación las modificaciones realizadas para arreglarlo:

```
# always loaded
'data': [
    'security/ir.model.access.csv',
    # Importamos las vistas
    'views/paciente.xml',
    'views/medico.xml',
    'views/consulta.xml',
],
```

Figura 45: Habilitamos el `'security/ir.model.access.csv'` del data.

Lo primero va a ser volver a nuestro `__manifest__.py` y en la parte de **data** teniamos comentado el archivo `'security/ir.model.access.csv'` lo descomentamos y ahora si nos iremos a modificar los permisos para poder ver nuesto módulo en Odoo.

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_paciente,paciente,model_paciente,base.group_user,1,1,1,1
access_medico,medico,model_medico,base.group_user,1,1,1,1
access_consulta,consulta,model_consulta,base.group_user,1,1,1,1
```

Figura 46: Actualizamos las reglas de acceso donde asignamos los permisos a los modelos.

Ahora en el `'security/ir.model.access.csv'` agregamos las reglas para permitir a los usuarios puedan `leer`, `crear`, `modificar`, `eliminar` los registros de **pacientes**, **médicos** y **consultas**. Permitiendonos así poder acceder y ver en el menú el módulo que creamos.

Con este pequeño error que nos dío solventado, vamos a enseñar el resultado de nuestro módulo de gestión hospital:

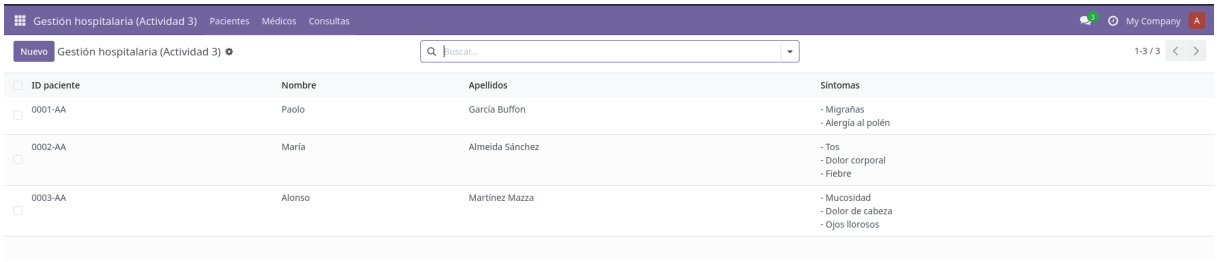



Figura 47: En esta captura tenemos la lista con todos los pacientes que creamos para nuestro módulo de gestión hospitalaria.

Como podemos apreciar en la captura estamos en la vista de pacientes donde tenemos tres pacientes, y vemos que los mostramos correctamente en el formato lista, ya que un hospital tiene muchos pacientes y el formato lista en mi opión no resulta muy útil, en la parte del menu tenemos las otras vistas de nuestro módulo que son médicos y consultas para poder movernos de vista.


Figura 48: En está captura tenemos el formulario para crear y editar los pacientes.

Como podemos ver ahora estamos en la vista de formulario de pacientes donde creamos o editamos un paciente y donde podemos ver que tenemos los campos: identificador, nombre, apellidos y los síntomas.

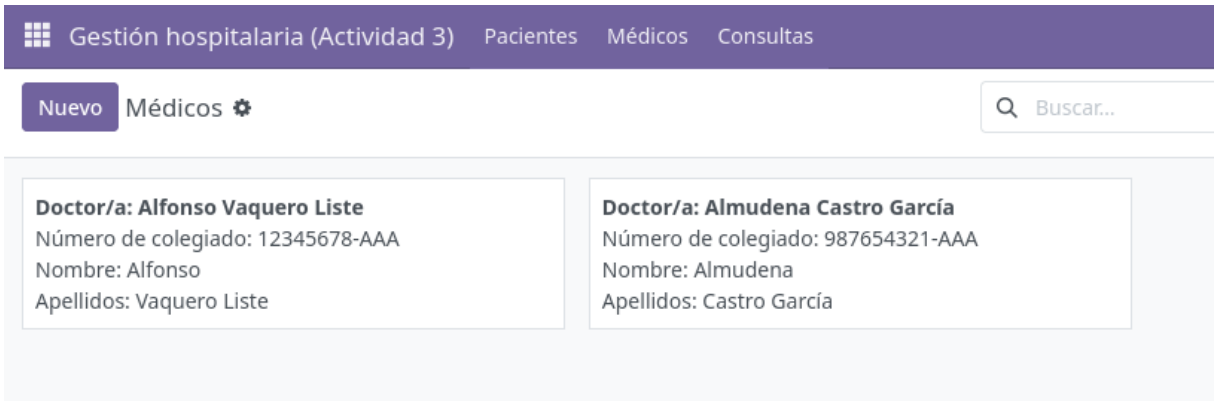



Figura 49: En esta captura mostramos los médicos que creamos para nuestro módulo en formato de tarjetas (kanban).

Esta es la vista de médicos donde mostramos los dos que creamos, y se visualizan en formato kanban ya que proporcionalmente hay muchos menos médicos que pacientes, me parecio mejor usar la vista kanban ya que mostraremos los médicos en tarjetas y visualmente se ve mejor.


Figura 50: En está captura tenemos el formulario para crear y editar los médicos.

Como podemos ver ahora estamos en la vista de formulario de médicos donde creamos o editamos un médico y donde podemos ver que tenemos los campos: número de colegiado, nombre y apellidos.

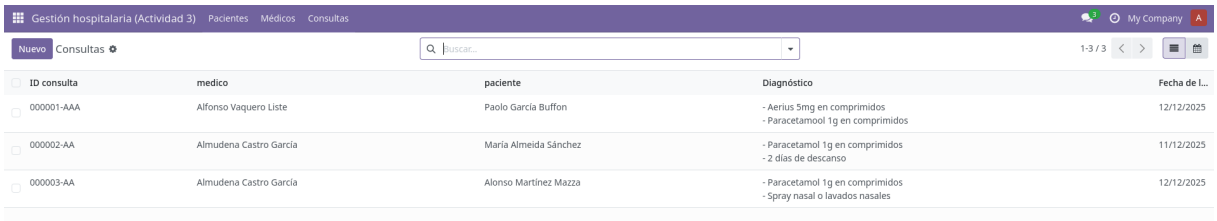



Figura 51: En esta captura tenemos la lista con todas las consultas que creamos en el módulo.

Esta es la vista de consultas donde mostramos los tres consultas que creamos, y se visualiza en formato lista y tambien en calendario ya que un hospital o un centro de salud se

producen muchas citas y poder verlas en lista y en formato calendario nos facilita la visualización.


Figura 52: En está captura tenemos el formulario para crear y editar las consultas médicas.

Como podemos ver ahora estamos en la vista de formulario de consultas donde creamos o editamos una consulta y donde podemos ver que tenemos los campos: identificador, médico que atiente, paciente, el diagnóstico y la fecha de la cita.

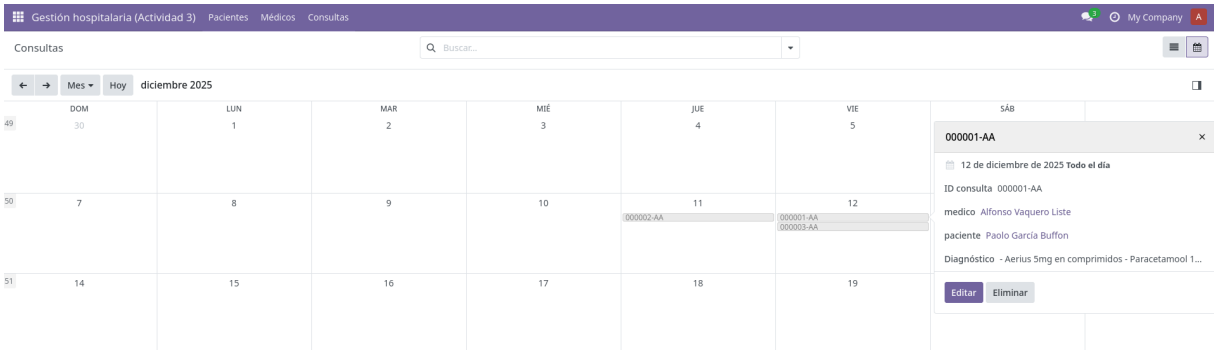



Figura 53: Aquí se ve la vista en formato calendario para ver en que día fue la consulta.

Aquí podemos ver la vista en formato calendario para mostrar las consultas, está la agregué adicionalmente porque se hace muy visual ver las citas en un calendario además de emplear el campo agregado adicionalmente `fecha_cita` que nos permite poder ver las consultas en este modo de visualización.

# 5. Actividad 04 - Ciclos formativos

# 6. Conclusión